Algorithmen und Datenstrukturen

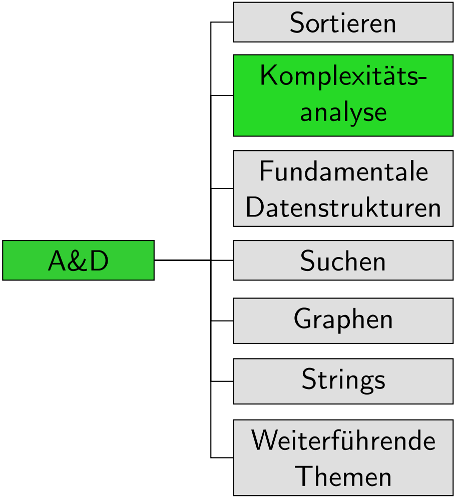
A6. Laufzeitanalyse: Top-Down-Mergesort und Landau-Symbole

Marcel Lüthi and Gabriele Röger

Universität Basel

27. Februar 2020

Inhalt dieser Veranstaltung



Was bisher geschah und wie es weiter geht

- Letztes Mal: sehr detaillierte Laufzeitanalyse für Selectionsort und Bottom-Up-Mergesort
- heute noch analoge Analyse für Top-Down-Mergesort als Beispiel eines rekursiven Divide-and-Conquer-Verfahrens
- danach Landau-Symbole für asymptotisches Laufzeitverhalten
- und die „schnelle" Laufzeitanalyse in der Praxis

Beispiel: Top-Down-Mergesort

Merge-Schritt-Ergebnis vom letzten Mal

```python
def merge(array, tmp, lo, mid, hi):
    i = lo
    j = mid + 1
    for k in range(lo, hi + 1):  # k = lo,...,hi
        if j > hi or (i <= mid and array[i] <= array[j]):
            tmp[k] = array[i]
            i += 1
        else:
            tmp[k] = array[j]
            j += 1
    for k in range(lo, hi + 1):  # k = lo,...,hi
        array[k] = tmp[k]
```

Theorem

Der Merge-Schritt hat lineare Laufzeit, d.h. es gibt Konstanten $c, c', n_0 > 0$, so dass für alle $n \geq n_0$: $cn \leq T(n) \leq c'n$.

Top-Down-Mergesort

```
1  def sort(array):
2      tmp = [0] * len(array)  # [0,...,0] with same size as array
3      sort_aux(array, tmp, 0, len(array) - 1)
4
5  def sort_aux(array, tmp, lo, hi):
6      if hi <= lo:
7          return
8      mid = lo + (hi - lo) // 2
9      sort_aux(array, tmp, lo, mid)
10     sort_aux(array, tmp, mid + 1, hi)
11     merge(array, tmp, lo, mid, hi)
```

Analyse für $m = \text{hi} - \text{lo} + 1$

c_0 für Zeile 6–7

c_1 für Zeile 6–8

$c_2 m$ für Merge-Schritt

Top-Down-Mergesort: Analyse I

Laufzeit `sort_aux`

- $T(m) = c_1 + 2T(m/2) + c_2 m$ für $m = 2^k, k \in \mathbb{N}_0$
- $T(1) = c_0$
- Rekursive Gleichung
- Wir suchen obere Schranke, die nur von m abhängt.

Top-Down-Mergesort: Analyse II

Betrachte $m = 2^k$ mit $k \in \mathbb{N}_{>0}$

$$T(m) = c_1 + 2T(m/2) + c_2 m$$

Top-Down-Mergesort: Analyse II

Betrachte $m = 2^k$ mit $k \in \mathbb{N}_{>0}$

$$\begin{aligned} T(m) &= c_1 + 2T(m/2) + c_2 m \\ &= c_1 + 2(c_1 + 2T(m/4) + c_2(m/2)) + c_2 m \end{aligned}$$

Top-Down-Mergesort: Analyse II

Betrachte $m = 2^k$ mit $k \in \mathbb{N}_{>0}$

$$
\begin{aligned}
T(m) &= c_1 + 2T(m/2) + c_2 m \\
&= c_1 + 2(c_1 + 2T(m/4) + c_2(m/2)) + c_2 m \\
&= c_1(1+2) + 2c_2 m + 4T(m/4)
\end{aligned}
$$

Top-Down-Mergesort: Analyse II

Betrachte $m = 2^k$ mit $k \in \mathbb{N}_{>0}$

$$\begin{aligned}
T(m) &= c_1 + 2T(m/2) + c_2 m \\
&= c_1 + 2(c_1 + 2T(m/4) + c_2(m/2)) + c_2 m \\
&= c_1(1+2) + 2c_2 m + 4T(m/4) \\
&= c_1(1+2) + 2c_2 m + 4(c_1 + 2T(m/8) + c_2(m/4))
\end{aligned}$$

Top-Down-Mergesort: Analyse II

Betrachte $m = 2^k$ mit $k \in \mathbb{N}_{>0}$

$$\begin{aligned}
T(m) &= c_1 + 2T(m/2) + c_2 m \\
&= c_1 + 2(c_1 + 2T(m/4) + c_2(m/2)) + c_2 m \\
&= c_1(1+2) + 2c_2 m + 4T(m/4) \\
&= c_1(1+2) + 2c_2 m + 4(c_1 + 2T(m/8) + c_2(m/4)) \\
&= c_1(1+2+4) + 3c_2 m + 8T(m/8)
\end{aligned}$$

Top-Down-Mergesort: Analyse II

Betrachte $m = 2^k$ mit $k \in \mathbb{N}_{>0}$

$$
\begin{aligned}
T(m) &= c_1 + 2T(m/2) + c_2 m \\
&= c_1 + 2(c_1 + 2T(m/4) + c_2(m/2)) + c_2 m \\
&= c_1(1+2) + 2c_2 m + 4T(m/4) \\
&= c_1(1+2) + 2c_2 m + 4(c_1 + 2T(m/8) + c_2(m/4)) \\
&= c_1(1+2+4) + 3c_2 m + 8T(m/8) \\
&= \ldots \\
&= c_1\left(\sum_{i=0}^{k-1} 2^i\right) + c_2 mk + c_0 2^k \\
&= c_1\left(\sum_{i=0}^{k-1} 2^i\right) + c_2 m \log_2 m + c_0 m \qquad (k = \log_2 m, 2^k = m)
\end{aligned}
$$

Top-Down-Mergesort: Analyse II

Betrachte $m = 2^k$ mit $k \in \mathbb{N}_{>0}$

$$\begin{aligned}
T(m) &= c_1 + 2T(m/2) + c_2 m \\
&= c_1 + 2(c_1 + 2T(m/4) + c_2(m/2)) + c_2 m \\
&= c_1(1+2) + 2c_2 m + 4T(m/4) \\
&= c_1(1+2) + 2c_2 m + 4(c_1 + 2T(m/8) + c_2(m/4)) \\
&= c_1(1+2+4) + 3c_2 m + 8T(m/8) \\
&= \dots \\
&= c_1\left(\sum_{i=0}^{k-1} 2^i\right) + c_2 mk + c_0 2^k \\
&= c_1\left(\sum_{i=0}^{k-1} 2^i\right) + c_2 m \log_2 m + c_0 m \qquad (k = \log_2 m, 2^k = m) \\
&\leq c_1 k 2^{k-1} + c_2 m \log_2 m + c_0 m \\
&\leq c_1 m \log_2 m + c_2 m \log_2 m + c_0 m \\
&\leq (c_0 + c_1 + c_2) m \log_2 m \qquad (\log_2 m = k \geq 1)
\end{aligned}$$

Top-Down-Mergesort: Analyse III

m keine Zweierpotenz? $2^{k-1} < m < 2^k$

$$
\begin{aligned}
T(m) &= c_1 + T(\lfloor m/2 \rfloor) + T(\lceil m/2 \rceil) + c_2 m \\
&\leq c_1 + 2T(2^k/2) + c_2 m \\
&\leq c 2^k \log_2 2^k \text{ für irgendein } c \\
&< 2cm \log_2(2m) \qquad (2^k < 2m, \text{ da } m > 2^{k-1}) \\
&= 2cm(\log_2 2 + \log_2 m) \\
&= 2cm(1 + \log_2 m) \leq 4cm \log_2 m \quad (1 \leq \log_2 m \text{ für } m \geq 2)
\end{aligned}
$$

Top-Down-Mergesort: Analyse III

m keine Zweierpotenz? $2^{k-1} < m < 2^k$

$$\begin{aligned}
T(m) &= c_1 + T(\lfloor m/2 \rfloor) + T(\lceil m/2 \rceil) + c_2 m \\
&\leq c_1 + 2T(2^k/2) + c_2 m \\
&\leq c2^k \log_2 2^k \text{ für irgendein } c \\
&< 2cm \log_2(2m) \qquad (2^k < 2m, \text{ da } m > 2^{k-1}) \\
&= 2cm(\log_2 2 + \log_2 m) \\
&= 2cm(1 + \log_2 m) \leq 4cm \log_2 m \quad (1 \leq \log_2 m \text{ für } m \geq 2)
\end{aligned}$$

Obere Schranke $c' m \log_2 m$ gilt allgemein (für irgendein c')

Top-Down-Mergesort: Analyse III

m keine Zweierpotenz? $2^{k-1} < m < 2^k$

$$
\begin{aligned}
T(m) &= c_1 + T(\lfloor m/2 \rfloor) + T(\lceil m/2 \rceil) + c_2 m \\
&\leq c_1 + 2T(2^k/2) + c_2 m \\
&\leq c 2^k \log_2 2^k \text{ für irgendein } c \\
&< 2cm \log_2(2m) \qquad (2^k < 2m, \text{ da } m > 2^{k-1}) \\
&= 2cm(\log_2 2 + \log_2 m) \\
&= 2cm(1 + \log_2 m) \leq 4cm \log_2 m \quad (1 \leq \log_2 m \text{ für } m \geq 2)
\end{aligned}
$$

Obere Schranke $c' m \log_2 m$ gilt allgemein (für irgendein c')

Untere Schranke?

$T(m) = \sum_{i=0}^{k-1} 2^i c_1 + c_2 m \log_2 m + c_0 m \geq c_2 m \log_2 m$

Untere Schranke $cm \log_2 m$ (für irgendein c)

Top-Down-Mergesort: Analyse IV

`sort`?

- Aufruf von `sort_aux` mit $m = n =$ Länge der Eingabe

Top-Down-Mergesort: Analyse IV

`sort`?

- Aufruf von `sort_aux` mit $m = n =$ Länge der Eingabe
- Anlegen/Kopieren von Array geht in linearer Zeit
  $\rightarrow$ kann durch Anpassung der Konstanten abgedeckt werden.

Top-Down-Mergesort: Analyse IV

`sort`?

- Aufruf von `sort_aux` mit $m = n =$ Länge der Eingabe
- Anlegen/Kopieren von Array geht in linearer Zeit
  $\rightarrow$ kann durch Anpassung der Konstanten abgedeckt werden.

Theorem

Top-Down-Mergesort hat leicht überlineare Laufzeit, d.h. es gibt Konstanten $c, c', n_0 > 0$, so dass für alle $n \geq n_0$, $cn \log_2 n \leq T(n) \leq c' n \log_2 n$.

Landau-Notation

Ergebnis für Mergesort

„Die Laufzeit von Mergesort wächst genauso schnell wie $n \log_2 n$."

Ergebnis für Mergesort

„Die Laufzeit von Mergesort wächst genauso schnell wie $n \log_2 n$."

Theorem

Mergesort hat leicht überlineare Laufzeit, d.h.
es gibt Konstanten $c, c', n_0 > 0$, so dass für alle $n \geq n_0$:
$c n \log_2 n \leq T(n) \leq c' n \log_2 n.$

Ergebnis für Mergesort

„Die Laufzeit von Mergesort wächst genauso schnell wie $n \log_2 n$."

Theorem

Mergesort hat leicht überlineare Laufzeit, d.h.
es gibt Konstanten $c, c', n_0 > 0$, so dass für alle $n \geq n_0$:
$cn \log_2 n \leq T(n) \leq c' n \log_2 n$.

- Wir haben Terme niedrigerer Ordnung (Konstanten und n) in der Abschätzung ignoriert bzw. verschwinden lassen.

Ergebnis für Mergesort

„Die Laufzeit von Mergesort wächst genauso schnell wie $n \log_2 n$."

Theorem

Mergesort hat leicht überlineare Laufzeit, d.h.
es gibt Konstanten $c, c', n_0 > 0$, *so dass für alle* $n \geq n_0$:
$cn \log_2 n \leq T(n) \leq c' n \log_2 n$.

- Wir haben Terme niedrigerer Ordnung (Konstanten und n) in der Abschätzung ignoriert bzw. verschwinden lassen.
- Wir haben uns nicht für die genauen Werte der Konstanten interessiert, es reicht, wenn irgendwelche passenden Konstanten existieren.

Ergebnis für Mergesort

„Die Laufzeit von Mergesort wächst genauso schnell wie $n \log_2 n$."

Theorem

Mergesort hat leicht überlineare Laufzeit, d.h.
es gibt Konstanten $c, c', n_0 > 0$, so dass für alle $n \geq n_0$:
$cn \log_2 n \leq T(n) \leq c' n \log_2 n$.

- Wir haben Terme niedrigerer Ordnung (Konstanten und n) in der Abschätzung ignoriert bzw. verschwinden lassen.
- Wir haben uns nicht für die genauen Werte der Konstanten interessiert, es reicht, wenn irgendwelche passenden Konstanten existieren.
- Die Laufzeit für kleine n ist nicht so wichtig.

Mehr bisherige Ergebnisse

Theorem

Der Merge-Schritt hat lineare Laufzeit, d.h. es gibt Konstanten $c, c', n_0 > 0$, so dass für alle $n \geq n_0$: $cn \leq T(n) \leq c'n$.

Theorem

Mergesort hat leicht überlineare Laufzeit, d.h.
es gibt Konstanten $c, c', n_0 > 0$, so dass für alle $n \geq n_0$:
$cn \log_2 n \leq T(n) \leq c'n \log_2 n$.

Theorem

Selectionsort hat quadratische Laufzeit, d.h. es gibt Konstanten $c > 0, c' > 0, n_0 > 0$, so dass für $n \geq n_0$: $cn^2 \leq T(n) \leq c'n^2$.

Mehr bisherige Ergebnisse

Theorem

Der Merge-Schritt hat lineare Laufzeit, d.h. es gibt Konstanten $c, c', n_0 > 0$, so dass für alle $n \geq n_0$: $cn \leq T(n) \leq c'n$.

Theorem

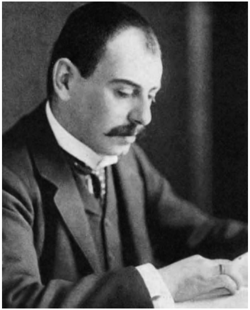
*Mergesort hat leicht überlineare Laufzeit, d.h.
es gibt Konstanten $c, c', n_0 > 0$, so dass für alle $n \geq n_0$:
$cn \log_2 n \leq T(n) \leq c'n \log_2 n$.*

Theorem

Selectionsort hat quadratische Laufzeit, d.h. es gibt Konstanten $c > 0, c' > 0, n_0 > 0$, so dass für $n \geq n_0$: $cn^2 \leq T(n) \leq c'n^2$.

Können wir das nicht irgendwie kompakter aufschreiben?

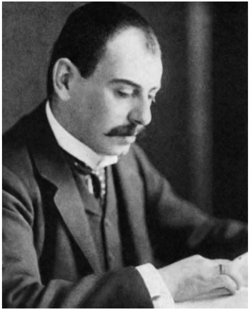
Edmund Landau



Edmund Landau

- deutscher Mathematiker (1877–1938)
- analytische Zahlentheorie
- kein Freund angewandter Mathematik

Edmund Landau



Edmund Landau

- deutscher Mathematiker (1877–1938)
- analytische Zahlentheorie
- kein Freund angewandter Mathematik

International: Bachmann–Landau-Notation auch nach Paul Gustav Heinrich Bachmann (deutscher Mathematiker)

Landau-Symbol Theta

Definition

Für eine Funktion $g : \mathbb{N} \to \mathbb{R}$ ist $\Theta(g)$ die Menge aller Funktionen $f : \mathbb{N} \to \mathbb{R}$, die genauso schnell wachsen wie g:

$$\Theta(g) = \{f \mid \exists c > 0 \; \exists c' > 0 \; \exists n_0 > 0 \; \forall n \geq n_0 : \\ c \cdot g(n) \leq f(n) \leq c' \cdot g(n)\}$$

Landau-Symbol Theta

Definition

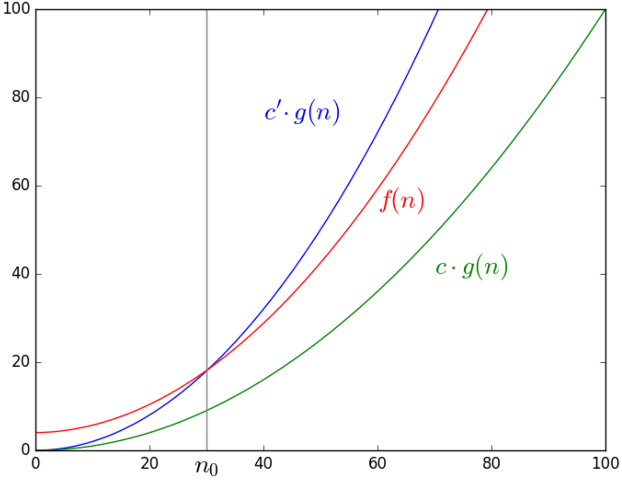
Für eine Funktion $g : \mathbb{N} \to \mathbb{R}$ ist $\Theta(g)$ die Menge aller Funktionen $f : \mathbb{N} \to \mathbb{R}$, die genauso schnell wachsen wie g:

$$\Theta(g) = \{f \mid \exists c > 0 \ \exists c' > 0 \ \exists n_0 > 0 \ \forall n \geq n_0 : \\ c \cdot g(n) \leq f(n) \leq c' \cdot g(n)\}$$

„Die Laufzeit von Mergesort ist in $\Theta(n \log_2 n)$."
oder auch
„Die Laufzeit von Mergesort ist $\Theta(n \log_2 n)$."

Landau-Symbol Theta: Illustration

$$f \in \Theta(g)$$



Mehr Landau-Symbole

- „f wächst nicht wesentlich schneller als g“

$$O(g) = \{f \mid \exists c > 0 \ \exists n_0 > 0 \ \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

Mehr Landau-Symbole

- „f wächst nicht wesentlich schneller als g“

$$O(g) = \{f \mid \exists c > 0\ \exists n_0 > 0\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

- O für „Ordnung“ der Funktion

Mehr Landau-Symbole

- „f wächst nicht wesentlich schneller als g“

$$O(g) = \{f \mid \exists c > 0 \ \exists n_0 > 0 \ \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

- O für „Ordnung“ der Funktion
- „f wächst nicht wesentlich langsamer als g“

$$\Omega(g) = \{f \mid \exists c > 0 \ \exists n_0 > 0 \ \forall n \geq n_0 : c \cdot g(n) \leq f(n)\}$$

Mehr Landau-Symbole

- „f wächst nicht wesentlich schneller als g“

$$O(g) = \{f \mid \exists c > 0\ \exists n_0 > 0\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

- O für „Ordnung“ der Funktion
- „f wächst nicht wesentlich langsamer als g“

$$\Omega(g) = \{f \mid \exists c > 0\ \exists n_0 > 0\ \forall n \geq n_0 : c \cdot g(n) \leq f(n)\}$$

- Es gilt $\Theta(g) = O(g) \cap \Omega(g)$.

Mehr Landau-Symbole

- „f wächst nicht wesentlich schneller als g“

$$O(g) = \{f \mid \exists c > 0\ \exists n_0 > 0\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

- O für „Ordnung“ der Funktion
- „f wächst nicht wesentlich langsamer als g“

$$\Omega(g) = \{f \mid \exists c > 0\ \exists n_0 > 0\ \forall n \geq n_0 : c \cdot g(n) \leq f(n)\}$$

- Es gilt $\Theta(g) = O(g) \cap \Omega(g)$.
- Es gilt $f \in \Omega(g)$ gdw. $g \in O(f)$.

Mehr Landau-Symbole

- „f wächst nicht wesentlich schneller als g“

$$O(g) = \{f \mid \exists c > 0 \; \exists n_0 > 0 \; \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

- O für „Ordnung“ der Funktion
- „f wächst nicht wesentlich langsamer als g“

$$\Omega(g) = \{f \mid \exists c > 0 \; \exists n_0 > 0 \; \forall n \geq n_0 : c \cdot g(n) \leq f(n)\}$$

- Es gilt $\Theta(g) = O(g) \cap \Omega(g)$.
- Es gilt $f \in \Omega(g)$ gdw. $g \in O(f)$.
- In der Informatik interessieren wir uns oft nur für die Begrenzung des Laufzeitwachstums nach oben: O statt Θ

Mehr Landau-Symbole

- „f wächst nicht wesentlich schneller als g“

$$O(g) = \{f \mid \exists c > 0\ \exists n_0 > 0\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

- O für „Ordnung“ der Funktion
- „f wächst nicht wesentlich langsamer als g“

$$\Omega(g) = \{f \mid \exists c > 0\ \exists n_0 > 0\ \forall n \geq n_0 : c \cdot g(n) \leq f(n)\}$$

- Es gilt $\Theta(g) = O(g) \cap \Omega(g)$.
- Es gilt $f \in \Omega(g)$ gdw. $g \in O(f)$.
- In der Informatik interessieren wir uns oft nur für die Begrenzung des Laufzeitwachstums nach oben: O statt Θ

Aussprache: Θ: Theta, Ω: Omega, O: Oh

Seltener benötigte Landau-Symbole

- „f wächst langsamer als g“

$$o(g) = \{f \mid \forall c > 0\ \exists n_0 > 0\ \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

Seltener benötigte Landau-Symbole

- „f wächst langsamer als g“

$$o(g) = \{f \mid \forall c > 0 \ \exists n_0 > 0 \ \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

- „f wächst schneller als g“

$$\omega(g) = \{f \mid \forall c > 0 \ \exists n_0 > 0 \ \forall n \geq n_0 : c \cdot g(n) \leq f(n)\}$$

Seltener benötigte Landau-Symbole

- „f wächst langsamer als g“

$$o(g) = \{f \mid \forall c > 0 \; \exists n_0 > 0 \; \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

- „f wächst schneller als g“

$$\omega(g) = \{f \mid \forall c > 0 \; \exists n_0 > 0 \; \forall n \geq n_0 : c \cdot g(n) \leq f(n)\}$$

Aussprache: ω: kleines Omega

Interessante Funktionsklassen

In aufsteigender Ordnung (abgesehen von allgemeinen n^k):

g	Wachstum
1	konstant
$\log n$	logarithmisch
n	linear
$n \log n$	leicht überlinear
n^2	quadratisch
n^3	kubisch
n^k	polynomiell (Konstante k)
2^n	exponentiell

Beispiele Θ

- Bei der Analyse interessiert nur der Term höchster Ordnung ($=$ am schnellsten wachsender Summand) einer Funktion.

Beispiele Θ

- Bei der Analyse interessiert nur der Term höchster Ordnung (= am schnellsten wachsender Summand) einer Funktion.
- Beispiele
  - $f_1(n) = 5n^2 + 3n - 9$
  - $f_2(n) = 3n \log_2 n + 2n^2$
  - $f_3(n) = 9n \log_2 n + n + 17$
  - $f_4(n) = 8$

Beispiele Θ

- Bei der Analyse interessiert nur der Term höchster Ordnung (= am schnellsten wachsender Summand) einer Funktion.
- Beispiele
  - $f_1(n) = 5n^2 + 3n - 9 \;\in \Theta(n^2)$
  - $f_2(n) = 3n \log_2 n + 2n^2$
  - $f_3(n) = 9n \log_2 n + n + 17$
  - $f_4(n) = 8$

Beispiele Θ

- Bei der Analyse interessiert nur der Term höchster Ordnung (= am schnellsten wachsender Summand) einer Funktion.
- Beispiele
  - $f_1(n) = 5n^2 + 3n - 9 \in \Theta(n^2)$
  - $f_2(n) = 3n \log_2 n + 2n^2 \in \Theta(n^2)$
  - $f_3(n) = 9n \log_2 n + n + 17$
  - $f_4(n) = 8$

Beispiele Θ

- Bei der Analyse interessiert nur der Term höchster Ordnung (= am schnellsten wachsender Summand) einer Funktion.
- Beispiele
  - $f_1(n) = 5n^2 + 3n - 9 \;\in \Theta(n^2)$
  - $f_2(n) = 3n \log_2 n + 2n^2 \;\in \Theta(n^2)$
  - $f_3(n) = 9n \log_2 n + n + 17 \;\in \Theta(n \log n)$
  - $f_4(n) = 8$

Beispiele Θ

- Bei der Analyse interessiert nur der Term höchster Ordnung (= am schnellsten wachsender Summand) einer Funktion.
- Beispiele
  - $f_1(n) = 5n^2 + 3n - 9 \;\in \Theta(n^2)$
  - $f_2(n) = 3n \log_2 n + 2n^2 \;\in \Theta(n^2)$
  - $f_3(n) = 9n \log_2 n + n + 17 \;\in \Theta(n \log n)$
  - $f_4(n) = 8 \;\in \Theta(1)$

Beispiele Gross-O

- Bei der Analyse interessiert nur der Term höchster Ordnung ($=$ am schnellsten wachsender Summand) einer Funktion.

Beispiele Gross-O

- Bei der Analyse interessiert nur der Term höchster Ordnung (= am schnellsten wachsender Summand) einer Funktion.
- Beispiele
  - $f_1(n) = 8n^2 - 3n - 9$
  - $f_2(n) = n^3 - 3n \log_2 n$
  - $f_3(n) = 3n \log_2 n + 1000n + 10^{200}$

Beispiele Gross-O

- Bei der Analyse interessiert nur der Term höchster Ordnung (= am schnellsten wachsender Summand) einer Funktion.
- Beispiele
  - $f_1(n) = 8n^2 - 3n - 9 \ \in O(n^2)$
  - $f_2(n) = n^3 - 3n \log_2 n$
  - $f_3(n) = 3n \log_2 n + 1000n + 10^{200}$

Beispiele Gross-O

- Bei der Analyse interessiert nur der Term höchster Ordnung (= am schnellsten wachsender Summand) einer Funktion.
- Beispiele
  - $f_1(n) = 8n^2 - 3n - 9 \ \in O(n^2)$
  - $f_2(n) = n^3 - 3n \log_2 n \ \in O(n^3)$
  - $f_3(n) = 3n \log_2 n + 1000n + 10^{200}$

Beispiele Gross-O

- Bei der Analyse interessiert nur der Term höchster Ordnung ($=$ am schnellsten wachsender Summand) einer Funktion.
- Beispiele
  - $f_1(n) = 8n^2 - 3n - 9 \; \in O(n^2)$
  - $f_2(n) = n^3 - 3n \log_2 n \; \in O(n^3)$
  - $f_3(n) = 3n \log_2 n + 1000n + 10^{200} \; \in O(n \log n)$

Beispiele Gross-O

- Bei der Analyse interessiert nur der Term höchster Ordnung (= am schnellsten wachsender Summand) einer Funktion.
- Beispiele
  - $f_1(n) = 8n^2 - 3n - 9 \;\in O(n^2)$
  - $f_2(n) = n^3 - 3n\log_2 n \;\in O(n^3)$
  - $f_3(n) = 3n\log_2 n + 1000n + 10^{200} \;\in O(n\log n)$
- Warum ist das so?

Zusammenhänge

Es gilt:

- $O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^k) \subset O(2^n)$ (für $k \geq 2$)

Zusammenhänge

Es gilt:

- $O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^k) \subset O(2^n)$ (für $k \geq 2$)
- $O(n^{k_1}) \subset O(n^{k_2})$ für $k_1 < k_2$
  z.B. $O(n^2) \subset O(n^3)$

Rechenregeln

- Produkt
  $f_1 \in O(g_1)$ und $f_2 \in O(g_2) \Rightarrow f_1 f_2 \in O(g_1 g_2)$

Rechenregeln

- Produkt
  $f_1 \in O(g_1)$ und $f_2 \in O(g_2) \Rightarrow f_1 f_2 \in O(g_1 g_2)$
- Summe
  $f_1 \in O(g_1)$ und $f_2 \in O(g_2) \Rightarrow f_1 + f_2 \in O(g_1 + g_2)$

Rechenregeln

- Produkt
  $f_1 \in O(g_1)$ und $f_2 \in O(g_2) \Rightarrow f_1 f_2 \in O(g_1 g_2)$
- Summe
  $f_1 \in O(g_1)$ und $f_2 \in O(g_2) \Rightarrow f_1 + f_2 \in O(g_1 + g_2)$
- Multiplikation mit Konstante
  $k > 0$ und $f \in O(g) \Rightarrow kf \in O(g)$
  $k > 0 \Rightarrow O(kg) = O(g)$

Grund für Beschränkung auf Term höchster Ordnung

Beispiel: $5n^3 + 2n \in O(n^3)$

- Wegen Regel bzgl. Multiplikation mit Konstante:
  - $5n^3 \in O(n^3)$
  - $2n \in O(n)$
- Wegen $O(n) \subset O(n^3)$ und $2n \in O(n)$:
  - $2n \in O(n^3)$
- Wegen Summenregel:
  - $5n^3 + 2n \in O(n^3 + n^3)$
- Mit Multiplikation mit Konstante (bei Klasse):
  - $5n^3 + 2n \in O(n^3)$

Anwendung

Schnelle O-Analyse für häufige Code-Konstrukte I

- konstante Operation

`var = 4`	$O(1)$

Schnelle O-Analyse für häufige Code-Konstrukte I

- konstante Operation

`var = 4`	$O(1)$

- Sequenz konstanter Operationen

`var1 = 4`	$O(1)$	$O(123 \cdot 1) = O(1)$
`var2 = 4`	$O(1)$	
`...`	$\cdots$	
`var123 = 4`	$O(1)$	

Schnelle O-Analyse für häufige Code-Konstrukte II

- Schleife

<table>
<tr><td rowspan="2"><pre>for i in range(n):
  res += i * m</pre></td><td>$O(n)$</td><td rowspan="2">$O(n \cdot 1) = O(n)$</td></tr>
<tr><td>$O(1)$</td></tr>
</table>

Schnelle O-Analyse für häufige Code-Konstrukte II

- Schleife

`for i in range(n):`	$O(n)$	$O(n \cdot 1) = O(n)$
`  res += i * m`	$O(1)$	

`for i in range(n):`	$O(n)$	$O(n)$	$O(n^2)$
`  for j in range(i):`	$O(n)$	$O(n)$	
`    res += i * (m - j)`	$O(1)$		

i hängt von n ab

Schnelle O-Analyse für häufige Code-Konstrukte III

- if-then-else

`if var < bound:`	$O(1)$	$O(1)$	$O(1 + \max\{1, n\})$ $= O(n)$
`  res += var`	$O(1)$	$O(1)$	
`else:`			
`  for i in range(n):`	$O(n)$	$O(n \cdot 1)$ $= O(n)$	
`    res += i * n`	$O(1)$		

Schnelle O-Analyse für häufige Code-Konstrukte III

- if-then-else

`if var < bound:`	$O(1)$	$O(1)$	$O(1 + \max\{1, n\})$ $= O(n)$
`  res += var`	$O(1)$	$O(1)$	
`else:`			
`  for i in range(n):`	$O(n)$	$O(n \cdot 1)$ $= O(n)$	
`    res += i * n`	$O(1)$		

Achtung: Kann zu unnötig hoher Abschätzung führen, wenn teurer Fall nur für kleine n auftritt (durch Konstante begrenzt).

Beispiel: Worst Case für Insertionsort

```python
def insertion_sort(array):
    n = len(array)
    for i in range(1, n):  # i = 1, ..., n - 1
        # move array[i] to the left until it is
        # at the correct position.
        for j in range(i, 0, -1):  # j = i, ..., 1
            if array[j] < array[j-1]:
                array[j], array[j-1] = array[j-1], array[j]
            else:
                break
```

Beispiel: Worst Case für Insertionsort

```python
def insertion_sort(array):
    n = len(array)
    for i in range(1, n):  # i = 1, ..., n - 1
        # move array[i] to the left until it is
        # at the correct position.
        for j in range(i, 0, -1):  # j = i, ..., 1
            if array[j] < array[j-1]:
                array[j], array[j-1] = array[j-1], array[j]
            else:
                break
```

- Worst case: `break`-Fall tritt nie ein.

Beispiel: Worst Case für Insertionsort

```python
def insertion_sort(array):
    n = len(array)
    for i in range(1, n):  # i = 1, ..., n - 1
        # move array[i] to the left until it is
        # at the correct position.
        for j in range(i, 0, -1):  # j = i, ..., 1
            if array[j] < array[j-1]:
                array[j], array[j-1] = array[j-1], array[j]
            else:
                break
```

- Worst case: `break`-Fall tritt nie ein.
- $O(1 + n \cdot n \cdot 1) = O(n^2)$

Beispiel: Worst Case für Insertionsort

```
def insertion_sort(array):
    n = len(array)
    for i in range(1, n):  # i = 1, ..., n - 1
        # move array[i] to the left until it is
        # at the correct position.
        for j in range(i, 0, -1):  # j = i, ..., 1
            if array[j] < array[j-1]:
                array[j], array[j-1] = array[j-1], array[j]
            else:
                break
```

- Worst case: `break`-Fall tritt nie ein.
- $O(1 + n \cdot n \cdot 1) = O(n^2)$
- Überschätzt?
  Nein, beide Schleifen haben jeweils $\Omega(n)$ Durchläufe.

Beispiel: Best Case für Insertionsort

```python
def insertion_sort(array):
    n = len(array)
    for i in range(1, n):  # i = 1, ..., n - 1
        # move array[i] to the left until it is
        # at the correct position.
        for j in range(i, 0, -1):  # j = i, ..., 1
            if array[j] < array[j-1]:
                array[j], array[j-1] = array[j-1], array[j]
            else:
                break
```

Beispiel: Best Case für Insertionsort

```python
def insertion_sort(array):
    n = len(array)
    for i in range(1, n):  # i = 1, ..., n - 1
        # move array[i] to the left until it is
        # at the correct position.
        for j in range(i, 0, -1):  # j = i, ..., 1
            if array[j] < array[j-1]:
                array[j], array[j-1] = array[j-1], array[j]
            else:
                break
```

- Best case: `break` jeweils direkt bei $j = i$

Beispiel: Best Case für Insertionsort

```python
def insertion_sort(array):
    n = len(array)
    for i in range(1, n):  # i = 1, ..., n - 1
        # move array[i] to the left until it is
        # at the correct position.
        for j in range(i, 0, -1):  # j = i, ..., 1
            if array[j] < array[j-1]:
                array[j], array[j-1] = array[j-1], array[j]
            else:
                break
```

- Best case: `break` jeweils direkt bei $j = i$
- $O(1 + n \cdot 1 \cdot 1) = O(n)$

Beispiel: Best Case für Insertionsort

```python
def insertion_sort(array):
    n = len(array)
    for i in range(1, n):  # i = 1, ..., n - 1
        # move array[i] to the left until it is
        # at the correct position.
        for j in range(i, 0, -1):  # j = i, ..., 1
            if array[j] < array[j-1]:
                array[j], array[j-1] = array[j-1], array[j]
            else:
                break
```

- Best case: `break` jeweils direkt bei $j = i$
- $O(1 + n \cdot 1 \cdot 1) = O(n)$
- Überschätzt?
  Nein, die äussere Schleifen hat $\Omega(n)$ Durchläufe.

Analyse Insertionsort mit Kostenmodell

```python
def insertion_sort(array):
    n = len(array)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if array[j] < array[j-1]:
                tmp = array[j]
                array[j] = array[j-1]
                array[j-1] = tmp
            else:
                break
```

- Best case: $n-1$ Schlüsselvergleiche, 0 Vertauschungen
- Worst case:
  $\sum_{i=1}^{n-1} i \in \Theta(n^2)$ Schlüsselvergleiche und Vertauschungen

Zusammenfassung

Zusammenfassung

- Mergesort hat auch in der Top-Down-Variante leicht überlineare Laufzeit.
- Mit Landau-Symbolen definiert man Klassen von Funktionen, die nicht schneller/nicht langsamer/... wachsen als eine Funktion g.
  - $O(g)$: Wachstum nicht schneller als g
  - $\Theta(g)$: Wachstum im Wesentlichen wie g
- Insertionsort hat
  - im besten Fall Laufzeit $\Theta(n)$
  - im schlechtesten Fall Laufzeit $\Theta(n^2)$